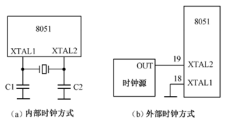
# 1. 8051 硬件结构

## 1.1 引脚功能

①**电源引脚**：VCC(5V)，VSS(GND)

②**时钟电路**：XTAL1：接外部晶振一端是片内振荡电路反相放大器的输入端。XTAL2：接外部晶振另一端是片内振荡电路反相放大器的输出端，从该引脚可输出频率为晶振频率的时钟信号。




③**复位引脚**：RST，输入 10ms 以上高电平脉冲，单片机复位。

④**其他控制引脚**：ALE：地址锁存允许信号输出端。有效时输出一个高脉冲。PSEN：外部 ROM 选通信号输出端，低电平有效。EA：内部、外部 ROM 选择信号输入端，低电平有效。

⑤**IO 引脚**：P0：需外接上拉电阻，第二功能是分时复用的 8 位数据线和低 8 位地址线；P1：单一功能 8 位准双向 I/O 口；P2：第一功能是普通 I/O 口；第二功能是高 8 位地址线；P3：第二功能如下表所列：

| | | | |
|---|---|---|---|
| P0 | RXD （串行口输入） | P4 | T0（定时器 0 计数输入） |
| P1 | TXD （串行口输出） | P5 | T1（定时器 1 计数输入） |
| P2 | $\overline{INT0}$（外部中断 0 输入） | P6 | WR（外部 RAM “写”选通） |
| P3 | $\overline{INT1}$（外部中断 1 输入） | P7 | RD（外部 RAM “读”选通） |

## 1.2 CPU 的结构与功能

CPU 由运算器和控制器两大部分组成。运算器是用来对数据进行算术运算和逻辑操作的执行部件；控制器是统一指挥和控制微控制器工作的部件。

①**控制器**：指令部件由 16 位程序计数器 PC(用于存放下一条取指指令的地址)、指令寄存器 IR（存放当前指令的操作码，等待译码）、指令译码器 ID（对当前指令操作码进行解析，完成指令规定的操作）组成。

②**运算器**：包含 ALU，累加器 ACC，PSW，A，B 寄存器等。

③**指令执行过程**：取指令->译码->执行

## 1.3 时钟和时序

①**节拍**：（时钟周期 P）单片机提供时钟信号的振荡源（OSC）的周期。

②**状态（S）**：2 个时钟周期，P1 和 P2

③**机器周期(Tw)**：12 个时钟周期。fOSC=6MHz->Tw=2us ； 12MHz -> 1us

④**指令周期**：1-4 个机器周期

## 1.4 储存结构

①**基本架构**：冯·诺依曼（也称普林斯顿）结构：程序存储器和数据存储器共用一个逻辑空间，且它们是统一编址的。哈佛结构：程序存储器和数据存储器分别编址。

②**8051 储存结构图**：哈佛结构

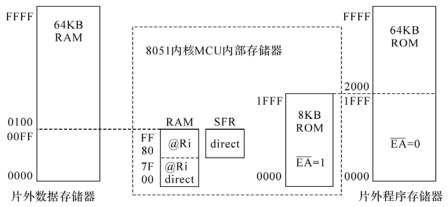



图 2-6　8051 MCU 的存储器结构与地址空间

③**程序存储器 ROM**：内部 8K：0-1FFFH；外部 64K：2000H-FFFFH；6 个入口：

| | | | | | |
|---|---|---|---|---|---|
| 复位 | 0x00 | 定时器 0 | 0x0B | 定时器 1 | 0x1B |
| 外部中断 0 | 0x03 | 外部中断 1 | 0x13 | 串行口中断 | 0x23 |

④**数据存储器 RAM**：内部 256B：0-0xFF；外部 64K：0-0xFFFF。

1. 工作寄存器：0-1FH，{RS1,RS0}={0,0}->0;{0,1}->1;{1,0}->2;{1,1}->3

2. 位寻址区：20H-2FH 共 16 个字节、128 位。

3. 用户 RAM 区：30H-FFH。其中 30H-7FH 可直接寻址，80H-FFH 仅可间接寻址。

4. 外部 RAM：仅可间接寻址。

⑤**SFR**：共 21 个离散分布于 80H-FFH，与内部 RAM 高位重叠，但 **SFR 直接寻址**！字节地址为 0 或 8 的既可字节寻址也可位寻址，额外注意 PSW 寄存器：

| 位地址 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| 位符号 | Cy | AC | F0 | RS1 | RS0 | OV | F1 | P |
| 注释 | Carry | Assistant Carry | Flag 0 | Register bank Selector bit 1 | Register bank Selector bit0 | Overflow | Flag 1 | Parity Flag |

A 中 **1 个数为奇数时 P 置 1**；符号加减法结果溢出：128~127 时 OV 置 1

# 2. 8051 指令系统

## 2.1 数据传送类

①**MOV**：注意右图中的 direct 包括 SFR。

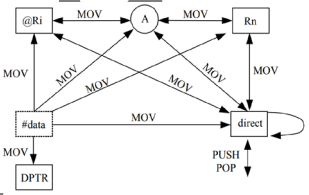
②**MOVX**：外部 RAM

@Ri←→A←→@DPTR

③**MOVC**：ROM

A←@A+DPTR

A←@A+PC（PC 为当前指令地址+1）




④**堆栈操作**：顺序！

PUSH direct：SP++,[SP]=[direct]；POP direct：[direct]=[SP]，SP--；

⑤**字节交换**：XCH A，Rn\direct\@Ri；半字节交换：XCHD A，@Ri；SWAP A

⑥**位传送**：MOV C,20H;MOV C,ACC.7;MOV P1.0,C

## 2.2 数据运算类

①**(带进位)加法指令**：ADD(C) A，#data\direct\@Ri\Rn 结果存在 A 中

②**带借位减法指令**：SUBB A，#data\direct\@Ri\Rn 结果存 A，注意清 Cy！

③**乘法指令**：MUL AB：将 AB 的 16 位乘积高 8 位存入 B，低 8 位存入 A

④**除法指令**：DIV AB：将 A/B 的商放入 A，余数放入 B，B=0 时 OV 置 1

⑤**加减 1 指令**：INC(DEC) A\Rn\@Ri\direct；INC DPTR；DPTR 减 1 程序如下：

CLR C|MOV A,DPL|SUBB A,#1|MOV DPL,A|MOV A,DPH|SUBB A,#0|MOV DPH,A

⑥**十进制调整**：DA A　只能用在 ADD 和 ADDC 之后！

## 2.3 逻辑操作类

①**逻辑与、或、异或**：ANL\ORL\XRL A，Rn\direct\@Ri\#data (A=A&\|\^…)；

ANL\ORL\XRL direct，A\#data

②**累加器清零和取反**：CLR A 累加器清零；CPL A 累加器逐位取反

③**循环移位指令**：RL\RR A 将 A 循环左右移 1 位；RLC\RRC A 将 Cy 放在 A7 和 A0 之间构成 9 位数据再进行循环左右移位。

## 2.4 控制转移类

注意下面的指令需要和**标号 lab** 配合使用，无需手动计算偏移量

①**无条件转移指令**：LJMP lab：3 字节指令，转移空间 64KB；AJMP lab：2 字节指令，可转移**同一个 2KB 空间**；SJMP lab:2 字节指令，下一条指令前后-128~127；JMP @A+DPTR 散转指令，跳转至 A+DPTR 地址，可实现 64KB 无条件转移。

②**条件转移指令**：JZ\JNZ lab：A 为 0\非 0 则跳转；CJNE 比较不相等则转移：

CJNE A，#data\direct，lab．CJNE Rn\@Rn，#data，lab

DJNZ：**先减 1** 若不为零则转移：DJNZ Rn\direct，lab

③**子程序调用和返回**：LCALL\ACALL lab 3 字节\2 字节子程序调用指令

调用过程：PC+3(LCALL,+2ACALL)，**PCL 入栈，PCH 入栈**，PC←lab

返回：RET(I) 子(中断)程序返回指令，调用过程：PCH 弹栈，PCL 弹栈

④**空操作**：NOP 消耗一个机器周期并使 PC 加 1

## 2.5 位操作指令

①**位传送指令**：MOV C,bit；MOV bit,C bit 位于位寻址区，两者中必须有 Cy!

②**位状态设置**：CLR\SETB\CPL bit\C：　清零\置位\翻转某一位

③**位逻辑运算**：ANL\ORL C,bit**/bit**：将 Cy 与 bit**bit 反**进行与\或操作

④**位转移指令**：JC\JNC lab：若 Cy 置位\为零则跳转至 lab；JB\JNB bit,lab：若 bit 置位\为零则跳转 lab，bit 不能是 Cy；JBC bit,lab 置位跳转并**清零**

## 2.6 指令周期与字节数

①**4 周期指令**：乘除法

②**2 周期指令**：MOVC;MOVX;PUSH;POP;含有 direct 但不含 A 的数据传送指令;DPTR 赋值和加一;含有 direct 的逻辑操作指令;除 NOP 所有控制转移指令;MOV bit,C;所有位转移指令

③**1 周期指令**：剩下所有指令

④**字节数判断**：无立即数、bit 和 direct 则单字节，含有几个则加几个字节

## 2.7 指令对 PSW 的影响

①**数据传送指令**除以 ACC 为目的操作数时对 P 有影响，其余均不影响。

②**算数运算指令**除 INC DEC 只影响 P，其余均影响所有标志位

③**逻辑操作指令**除 ACC 目的操作数影响 P，带进位循环影响 Cy，其余均不影响

④**控制转移指令**除 CJNE 影响 Cy，其余均不影响

⑤**位操作指令**除了对 Cy 操作时影响 Cy，其余均不影响

# 3. 汇编程序与 C 程序设计

## 3.1 汇编程序伪指令

①**起始汇编**：ORG nn nn 为 16 位常数，表示接下来的内容位于 ROM 的 nn 处

②**赋值**：X EQU Y：X 为字符名，Y 可以是标号、常数、SFR、**位地址**等

③**定义字节**：例如 DB 73H,04,100,-2,”ABC”

④**定义字**：例如 DW 100H,3456H,… 按**大端规则**分别存入 01H 00H 34H 56H…

⑤**位地址赋值**：例如 B1 BIT ACC.0\00H\P3.1，也可以替换为 EQU

⑥**保留储存空间**：例如 BASE: DS 100H 表示从 BASE 处保留 100H 个单元备用

## 3.2 汇编子程序设计举例：

①**普中科技延时 200ms（12MHz）**：更小的延时可以选取更少的循环层数

```
DELAY200MS:             2us(LCALL)
        PUSH 30H        2us(前面可加 nop 调整奇偶)
        PUSH 31H        2us
        PUSH 32H        2us
        MOV 30H,#2  (Z) 2us
        MOV 31H,#134(Y) 2us
        MOV 32H,#13 (X) 2us
NEXT:                   (循环层数 L=3)
        DJNZ 32H,NEXT   13*2
        DJNZ 31H,NEXT   (256*2+2)*(134-1)+2
        DJNZ 30H,NEXT   ((256*2+2)*256+2)*(2-1)+2
        POP 32H         2us
        POP 31H         2us
        POP 30H         2us
        RET             2us
```

Delay=X*2+(256*2+2)*(Y-1)+2+((256*2+2)*256+2)*(X-1)+2+2*3+L+4

= 2*X+514*(Y-1)+131586*(Z-1)+6*L+8 （可推广）

②冒泡排序

## 3.3 C51 程序设计

①**C51 拓展的数据类型**：

bit：申明位变量，例如 bit flag=0;

sbit：申明 sfr 或 bdata 中的某一位例如 sbit P11=P1^1;

sfr\sfr16：sfr P0=0x80 sfr16 DPTR=0x82；注意’=’后面应该是地址常数

②**储存类型**：data：0-127 可直接寻址的内部 ram；bdata：可位寻址区，例如 unsigned char bdata sta；sbit RX_DR = sta^6；idata 0-255 可间接寻址内部 ram；xdata：全 64K 外部 rom；pdata：低 256B 外部 ram；code：64K rom

③**存储模式**：SMALL：默认 data，栈位于片内（最快）；COMPACT：默认 pdata，使用@R0 或 R1 访问，栈位于片内；LARGE：默认 xdata，使用 DPTR 访问(最慢)

④**指针**：数据类型<指向数据的存储类型> * [指针变量的存储类型] 指针名；

<>[]可缺省由编译模式指定，未指定<>时称为通用指针，否则为特殊指针

例如：int xdata * data a;指向外部 ram 的 int，指针位于直接寻址 ram 区

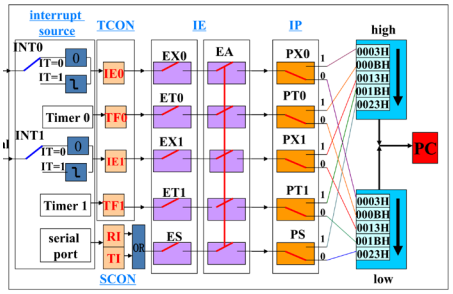
应用：将外部 ram0x1000 开始 10 个 B 的数复制到内部 ram0x20 开始处

```
unsigned char data i,*dpt=0x20; unsinged char xdata *xpt=0x1000;
for(i = 0;i < 10;i++) *(dpt+i) = *(xpt+i);
```

⑤**函数定义**：[return_type]　funcname([args])[{small|compact|large}][reentrant][interrupt n][using n]　需要注意函数若需递归必须使用 reentrant 提示编译器使用栈传参，否则默认使用寄存器传参导致错误

# 4. 中断系统

## 4.1 中断系统结构




## 4.2 中断的控制

①**中断标志清除**：定时器溢出并开启中断时 TF0、TF1 自动清零，不开启中断则需软件清零；当下降沿触发时，IE0、IE1 自动清零，低电平触发时保持至高电平时清零；串行口中断标志均需软件清零。

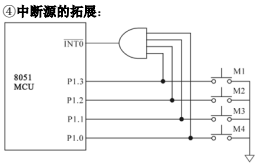
②**中断程序初始化**：按照结构图逐步设置即可

例：设置 INT0 电平高优先级触发 SETB EA；SETB EX0；SETB PX0；CLR IT0

③**中断服务程序**：void INT(void) interrupt n{}　(n 与自然优先级顺序一致)

使用汇编时应注意保护现场和使用 RETI

④**中断源的拓展**：

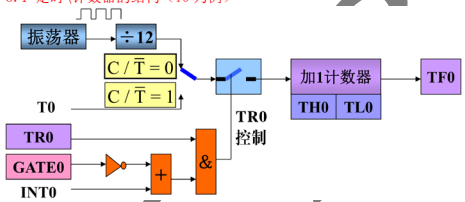



⑤**相关 SFR 结构**：


（从上至下 IE TCON IP）

# 5. 定时\计数器

## 5.1 定时\计数器的结构（T0 为例）




## 5.2 定时\计数器的控制

①**方式寄存器 TMOD**：低位不是 0 或 8，只能整体赋值！

| 89H | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| | GATE | C/$\overline{T}$ | M1 | M0 | GATE | C/$\overline{T}$ | M1 | M0 |
| | ←——T1——→ | | | | ←——T0——→ | | | |

1.GATE：当 GATE0=1 时，定时器 0 仅当 P3.2$\overline{INT0}$为高电平、TR0=1 时启动。

当 GATE0=0 时，当 TR0=1 时定时器即开始工作，可以实现**脉宽测量**。

2.C/$\overline{T}$：当 C/$\overline{T}$=0 时定时方式，=1 时计数方式。外部脉冲频率应小于 fosc/24

3.M1\M0：共四种工作方式。当 M1\M0={0,1}时为方式 2，16 位计数器，M1\M0={1,0}时为方式 2，自动重装 8 位计数器。

②**控制寄存器 TCON**：可以位寻址！

| 88H | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| | TF1 | TR1 | TF0 | TR0 | IE1 | IT0 | IE0 | IT0 |

1.TF0：溢出标志，T0 溢出时置 1，中断方式下硬件自动清零

2.TR0：启停控制，当 GATE=0 时 TR0=1 时 T0 工作，当 GATE=1 时需要配合 INT0

③**TH、TL**：定时器初值确定（假设 fosc=12MHz，即单位时长为 1us，其他同理）

对方式 1：M=2^16=65536，初值 X，定时时长 t=M-X(*1us@12MHz *2us@6MHz)

对方式 2：M=2^8=256，初值 X，定时时长 t=M-X (*1us@12MHz *2us@6MHz)

对两种方式：t=(M-X)*机器周期 ⟹ X=M-t/机器周期

## 5.3 三种典型应用

①**定时方式**：fosc=12MHz，T1 定时，P1.0 输出周期 2ms 的方波

查询方式：X=65536-1000(1ms)=0xFC18

```
        MOV   TMOD,#10H   ;设置T1为方式1
        SETB  TR1         ;启动T1定时
LOOP:   MOV   TH1,#0FCH
        MOV   TL1#18H     ;装入定时初值
        JNB   TF1,$       ;等待溢出，若TF1=0，则继续查询等待
        CPL   P1.0        ;P1.0状态翻转，输出方波
        CLR   TF1         ;查询方式，TF1需要软件清0
        SJMP  LOOP        ;重复循环
```

中断方式：

```
        ORG   0000H
        LJMP  MAIN        ;跳转到主程序
        ORG   001BH       ;T1中断入口
        LJMP  T1SUB       ;转T1中断服务程序
        ORG   0100H
MAIN:   MOV   TMOD, #10H  ;设置T1方式1
        MOV   TH1,#0FCH
        MOV   TL1,#18H    ;设置定时初值
        SETB  EA          ;CPU中断允许
        SETB  ET1         ;T1中断允许
        SETB  TR1         ;启动T1
        SJMP  $           ;模拟主程序
```

②**计数方式**：

要求：每计数 100 个脉冲中断一次。

分析：计数模式下 C/$\overline{T}$须置为 1，若采用方式 2 则 X=256-100=0x9C

```
ORG     0800H
T1SUB:  MOV   TH1,#0FCH
        MOV   TL1,#18H    ;重装载定时初值
        CPL   P1.0        ;P1.0状态翻转，输出方波
```

②**计数方式**：要求分析见右上

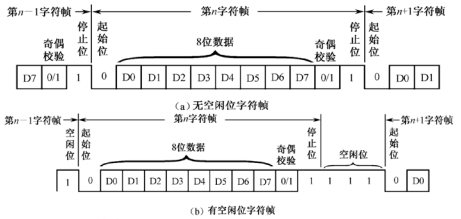
```
sbit  P10=P1^0;
int  Total;
void main()
{
    P10=0;    //P1.0低电平
    Total=0;  //箱数清0
    TMOD=6;  //设置T0工作方式
    TH0=0x9C; //设置重装载初值
    TL0=0x9C; //设置计数初值
    EA=1;
    ET0=1;
    TR0=1;
     while (1);
}
```

```
void timer0() interrupt 1 using 0
{
    int i;
    Total++;              //箱数加1
    P10=1;                //输出包装机打包信号
    for (i=0;i<10;i++);   //高脉冲保持一定宽度
    P10=0;                //结束脉冲输出
}
```

③**脉宽测量**：利用 T1，测量 INT1 引脚上的高脉冲宽度。设 fosc=12MHz，高脉冲宽度小于 65535us。设置 C/$\overline{T}$=0，工作方式 1，GATE=1，初值 X=0。当 INT1 低电平期间设置 TR1=1，待 INT1 再回到低电平时置 TR1=0，完成一次测量。

```
MOV   TMOD,#90H     ;设置方式1、GATE=1
CLR   TR1
MOV   TL1,#0
MOV   TH1,#0
JB    P3.3, $       ;判断INT1是否为低电平（需要在低电平时，令TR1=1）
SETB  TR1           ;在INT1=0时，置TR1=1，使计数器的启停受控于INT1
JNB   P3.3, $       ;等待INT1变高电平，使计数器开始计，开始脉宽测量
JB    P3.3, $       ;等INT1变为低电平，使计数器停止计数，结束脉宽测量
CLR   TR1           ;置TR1=0，为下次测量做准备
MOV   A, TL1        ;读取16位计数器累计的机器周期数，高低8位分别送B、A寄存器
MOV    B, TH1
SJMP  $             ;一次测量结束
```

## 6. 串行通信 UART




6.1 UART 通信帧格式

6.2 相关寄存器

①**SCON（98H）**：D7-> |SM0|SM1|SM2|REN|TB8|RB8|TI|RI| <-D0

{SM0,SM1}控制工作方式：{0,0}方式 0，{0,1}方式 1，{1,0}方式 2，{1,1}方式 3；

SM2：在**方式 2、3 中**若 **SM2=1** 且接收到 **RB8=0** 时不激活 RI(可多机通信)；

REN：REN=1 允许接收； TB8\RB8：发送\接收的第 9 位，可用作校验等；

TI\RI：发送\接收结束中断标志，需要软件清零。

②**PCON（87H）**：D7-> |**SMOD**|--|--|--|GF1|GF0|PD|IDL| <-D0

方式 1-3 下当 SMOD=1 时波特率加倍

③**SBUF（99H）**：发送和接收缓冲器，物理上相互独立

6.3 工作方式

①**方式 0**：串行数据由 RXD 输入输出，TXD 输出同步移位脉冲，常用于串并互转

②**方式 1**：10 位异步，起始位(=0 自动插入)+8 位数据+停止位(=1 自动插入)

②**方式 2\3**：11 位异步，停止位前插入 TB8\RB8，波特率不同，注意 SM2=1 时！

6.4 波特率 注意 SMOD 位于 PCON，取 0 或 1

①**方式 0**：固定为 fosc/12，即机器周期；②**方式 2**：2^SMOD * fosc/64

②**方式 1\3**：波特率=2^SMOD/32*t（T1 溢出周期）代入 t=12/fosc*(256-X)得 T1 初值 X = 256-fosc*(SMOD+1)/(384*波特率)

6.5 串行通信实例

要求：发送方将存放在 50H-5FH16 个数据发送给接收方，采用方式 3 偶校验，波特率为 1200bps，fosc=11.0592MHz

分析：SM0=1,SM1=1,定时器 1 工作在方式 2，初值 0xE8

发送程序：

```
#include<reg51.h>
char idata *p = 0x50;
int i;
void main(){
    SM0=SM1=1;
    TMOD=0x20;
    TL1=TH1=0xE8;
    TR1=1;
    for(i=0;i<16;i++){
        ACC=*(p+i);
        TB8=P;//汇编需用 Cy
        SBUF=*(p+i);
        while(TI==0);//等待
        TI=0;
    }
}
```

接收程序：

```
#include<reg51.h>
char idata *p = 0x50;
int i,error=0;
void main(){
    SM0=SM1=1;REN=1;
    TMOD=0x20;
    TL1=TH1=0xE8;
    TR1=1;
    for(i=0;i<16;i++){
        while(RI==0);//等待
        RI=0;
        ACC=SBUF;
        if(P==RB8) *(p+i)=ACC;
        else {error=1;break;}
    }
}
```

6.6 多机通信（需要 RS485 总线网络）

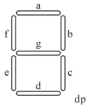
一主多从位于同一总线，工作在方式 2 或 3。初始时从机 SM2=1，主机置 TB8=1 并广播寻址，从机若收到本机地址则置 SM2=0 并应答，主机收到应答后置 TB8=0 并开始一对一通信。通信结束后从机重置 SM2=1，主机可再次寻址通信。

## 7. 人机接口

7.1 矩阵式键盘

①**行扫描法**：行线为输出，列线为输入，当无按键按下时列输入全为 1。将某一行输出为 0 读取列线值，若某一列为 0 则表明该行列交叉处按键被按下。

②**线路反转法**：先行输出全 0，读入列线电平；再列输出全 0，读入行线电平；将两次读入的电平值合成特征码(例如 E\B\D\7 组合)即可查找到对应的按键。




7.2 LED 段码管

编号如右图：共阳：低电平点亮；共阴：高电平点亮；审题！

## 8. 嵌入式系统概论

8.1 嵌入式系统定义和分类：

①**国内定义**：以应用为中心、以计算机技术为基础，软硬件可裁减，适用于应用系统的、对功能、可靠性、成本、体积、功耗有严格要求的专用计算机系统。

②**从复杂程度分类**：初级：MSC51 为代表的 8 位 MCU；中级以 ARM7/Cortex-M3/4 为代表的 32 位机系统；高级：以 Cortex-A8/15 等为代表的 32/64 位机系统。

③**处理器分类**：CISC：Intel；RISC：PowerPc、MIPS、ARM

经典 RISC 的特点：大的、统一的寄存器文件；装载/保存结构，数据处理操作只针对寄存器的内容，而不直接对存储器进行操作；简单的寻址模式； 统一和固定长度的指令域，简化了指令的译码；

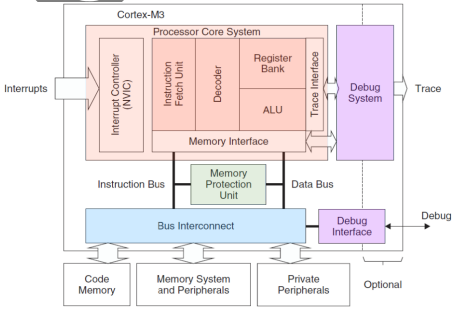
8.2 CM3 现状：

LPC17000(NXP),LM3S(TI),STM32(ST),SAM3U(Atmel)为四大厂商采用 M3 处理器设计出的芯片，区别在片上存储器容量、集成外设、功能模块


8.3 嵌入性系统组成模型：

## 9. CM3 架构与指令系统

9.1 Cortex-M3 内核结构：




9.2 Cortex-M3 三级流水线：

M3 处理器使用了流水线+分支预测，使几个操作同时进行，例如 ARM\Thumb2：PC-8:执行;PC-4:译码;PC:取指； Thumb:PC-4:执行;PC-2:译码;PC:取指

9.3 Thumb-2 指令集：

可以实现 ARM 指令的所有功能。代码性能达到了纯 ARM 代码性能的 98%，大小仅有其 74%。代码密度比现有的 Thumb 指令集更高。**CM3 不支持 ARM 指令集**。

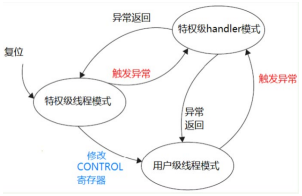
9.4 两种工作模式

①**处理模式**：用于 ISR

②**线程模式**：普通程序

③**特权级**：可以访问所有范围的存储器，可以执行所有的指令；

④**用户级**：需执行一条系统调用指令由异常服务例程修改控制寄存器才能重新进入特权级。




9.5 存储器保护单元 MPU：

保护内存的一个组件，方法是通过函数访问内存。MPU 为以下操作提供完整的支持：保护区域、重叠保护区区域、访问权限、将存储器属性输出到系统；MPU 可以用于:强制执行特权原则、分离程序、强制执行访问原则。

9.6 内部寄存器：

R0-R15 共 16 个，其中 R0-R12 为通用寄存器；R13 作为堆栈指针 SP，SP 有两个分别为主堆栈指针 MSP 和进程堆栈指针 PSP，但同时只能有一个可以使用；R14 是连接寄存器 LR，R15 是程序计数器 PC。

9.7 中断机制：

①**特点**：中断优先级可动态重新设置；中断数目可配置为 1～240；中断优先级的数目可配置为 1-8 位(1-256 级)；处理模式和线程模式具有独立的堆栈和特权等级；使用 C/C++标准的调用规范

②**中断过程**：硬件压栈→读向量表→从向量表中读 SP→更新 PC→加载流水线→更新 LR；以上步骤由硬件自动完成，仅需 12 个时钟。

③**优先级**：通过对中断优先级寄存器的 8 位 PRI_N 区执行写操作，将中断的优先级指定为 0-255，0 最高，255 最低。优先级的设置对复位，NMI 和硬故障无效，它们的优先级始终比外部中断要高。如果两个或更多的中断指定了相同的优先级，则由它们的硬件优先级来决定处理器对它们进行处理时的顺序。

④**占先**：如果新的异常比当前的异常优先级更高，则打断当前响应，产生嵌套

⑤**末尾迁帧**：为加速响应，若当前 ISR 结束时存在比待返回 ISR 或线程优先级更高的挂起中断，则会跳过出栈操作转而将控制权让给被挂起的高优先级中断。

9.8 复位功能：系统复位、电源复位和后备域复位。

9.9 内存对齐：

硬件支持的字、半字、字节分别占用 4、2、1 个字节，对齐的字和半字操作指令分别需要其地址为 4 或 2 的整数倍，否则会发生非对齐操作。

9.10 调试接口：CM3 处理器的调试接口有 2 种：JTAG(6 线)； SWD(2 线)

## 10. STM32F103 的功能部件

10.1 GPIO

①**GPIO 工作模式配置**：GPIO_Mode_XXX XXX 为模式名，分别为：AIN 模拟输入、IN_FLOATING 浮空输入（**复位后默认**）、IPD 下拉输入、IPU 上拉输入、Out_OD 开漏输出、Out_PP 推挽输出、AF_OD 复用开漏输出、AF_PP 复用推挽输出。

②**GPIO 寄存器**：

32 位配置寄存器，设置端口工作模式：GPIOx_CRL(低 8 口)，GPIOx_CRH(高 8 口)

32 位数据寄存器，输入和输出状态位于低 16 位：GPIOx_IDR，GPIOx_ODR

32 位置位/复位寄存器：低 16 位置 1 输出高电平，高 16 位相反：GPIOx_BSRR

16 位复位寄存器，置 1 即清零：GPIOx_BRR

32 位锁定寄存器，不常用：GPIOx_LCKR

③**重要函数**：

GPIO_Init GPIO_ReadInputDataBit GPIO_ReadInputData

GPIO_ReadOutputDataBit GPIO_ReadOutputData GPIO_SetBits

GPIO_ResetBits GPIO_WriteBit GPIO_Write

③**初始化例程**：

```
void LED_Init(void){
        GPIO_InitTypeDef GPIO_InitStructure;
        GPIO_InitStructure.GPIO_Pin = PIN_LED;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
        RCC_APB2PeriphClockCmd(RCC_LED, ENABLE);
        GPIO_Init(GPIO_LED, &GPIO_InitStructure);
}
```

④**应用例程**：

```
void LED_Sets(uint8_t data){
        uint16_t setValue;
        setValue = GPIO_ReadOutputData(GPIO_LED);
        // setValue = GPIOx->ODR
        setValue &= 0x00ff;
        setValue |= (uint16_t)data << 8;
        GPIO_Write(GPIO_LED,setValue);
        // GPIOx->ODR = setValue
}
```

10.2 定时\计数器

①**特点**：四个通用 16 位定时器 TIM2-5 具有 4 个比较、扑获通道；计数模式：上升、下降、上升或下降；T1 为增强型带互补输出紧急停止等功能可用于电机控制；中断产生条件：定时器溢出、比较值相等、扑获引脚有指定的跳变；

②**IRQ 例程**：

```
void TIM5_IRQHandler(void)
{
        if(TIM_GetITStatus(TIM5,TIM_IT_Update)){
                TIM_ClearITPendingBit(TIM5,TIM_IT_Update);
                led2=!led2;
        }
}
```

10.3 UART

①**例程**：中断接收 1 个字节，马上发送该字节

```
void USART1_IRQHandler(void) {
    u8 r;
    if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET) {
        r=USART_ReceiveData(USART1);;
        USART_SendData(USART1,r);
        while(USART_GetFlagStatus(USART1,USART_FLAG_TC)!=SET);
    }
    USART_ClearFlag(USART1,USART_FLAG_TC);
}
```

②**例程**：USART 初始化

```
USART_InitTypeDef USART_InitStructure;
USART_InitStructure.USART_BaudRate = 9600;//波特率设置
USART_InitStructure.USART_WordLength=USART_WordLength_8b;//字长为 8 位
USART_InitStructure.USART_StopBits = USART_StopBits_1;//一个停止位
USART_InitStructure.USART_Parity = USART_Parity_No;//无奇偶校验位
USART_InitStructure.USART_HardwareFlowControl=
USART_HardwareFlowControl_None;//无硬件数据流控制
USART_InitStructure.USART_Mode=USART_Mode_Rx|USART_Mode_Tx;//收发模式
USART_Init(USART1, &USART_InitStructure); //初始化串口 1
USART_Cmd(USART1, ENABLE);  //使能串口 1
USART_ClearFlag(USART1, USART_FLAG_TC);//清除待处理标志位
 USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);//开启相关中断
```

④**例程**：主程序

```
int main(){
    RCCINIT(); //系统时钟初始化
    GPIOINIT();
    USARTINIT();
    NVICINIT();//中断模式初始化
    while(1);//等待中断
}
```

10.4 A/D 转换器

①**参数**:2 个 12 位的逐次比较型 ADC,ADC 有 16 个外部通道,转换时间 1~1.5us,

②**工作方式**：单次转换模式、连续转换模式、扫描转换模式。启动 A/D 转换可由软件命令、定时器(TIM1)产生的事件、外部触发和 DMA 触发；A/D 转换结束后自动产生中断。CPU 通过查询状态位、中断响应、DMA 方式获取 A/D 值。

③**例程**：

```
int main()
{
    u32 ad=0;u8 i;
    RCCINIT_ADC();      //初始化 ADC 的系统时钟
    GPIOINIT_ADC();     //初始化 ADC 的端口配置
    ADCINIT_ADC();
    while(1){
      ad=0;
      for(i=0;i<50;i++){
        ADC_SoftwareStartConvCmd(ADC1, ENABLE);
        while(!ADC_GetFlagStatus(ADC1,ADC_FLAG_EOC));//等待转换结束
        ad=ad+ADC_GetConversionValue(ADC1);//累加转换结果
      }
      ad=ad/50;//平均后的结果
      delay_ms(1000);
    }
}
```

10.5 D/A 转换器

①**特点**：STM32F103C8 中无 DAC；2 个独立的 12 位 D/A，每个 D/A 有一个电压信号输出端；能产生三角波、随机噪声波；每个 DAC 具有 DMA 传送能力；带外部触发信号，启动 D/A 转换；外接参考电压 VREF+（与 ADC 共用）输入，可提高 DAC 分辨率；输出电压计算： Vout = DA /4096* (VREF+ — VREF-)

②**工作模式**：单通道模式和双通道模式；双通道模式中，2 个 DAC 可以独立运行，或同时转换；每个 DAC 可选择 8 位或 12 位转换模式；在 12 位转换模式中，数据可选择左对齐，或右对齐。

③**例程**：

```
int main()
{
  u8 i;
  float da;
  RCCINIT();
  GPIOINIT();
  NVICINIT();
  USARTINIT();
  DACINIT();
  while(1)
  {
    da=0;
    for(i=0;i<=10;i++){
      da=i*400;
      DAC_SetChannel1Data(DAC_Align_12b_R,da);//12 位右对齐 PA4 端口输出
      printf("da=%fv\n",3.3*da/4096);
      delayms(5000); //间隔 5 秒输出一个电压
    }
  }
}
```

CSDN @Cc98